2023牛客暑期多校训练营4题解

A. Bobo String Construction

我们的做法依赖于以下结论。

Theorem A.1. s取全0串或者全1中，至少有一者满足题述条件(即s在$t+s+t$中仅出现两次)。

为了证明Theorem A.1,我们需要用到字符串的border的定义。

Definition A.2 (border)**.** 一个字符串s是一个字符串t的border当且仅当s同时是t的一个真前缀以及t的一个真后缀。

我们接下来证明Theorem A.1。(更严格的证明是用数学归纳法，但这样"递归"的不那么严格的证明可能更好读一些)

Proof of Theorem A.1. 如果t不存在border, 命题显然成立。否则假设t'是t的最长border. 我们要满足的要求即为$t'+s+t'$中不出现t。如果$t'+s+t'$长度与t相同，两者只有一种匹配方式，故全0和全1中至少有一者满足条件。如果t长度更长，那么无论s是什么都可以满足条件;如果t长度更短，如果$t'+s+t'$中出现t，这个t的前缀t'与后缀t'不同时是$t'+s+t'$的前缀t'与后缀t', 所以此时$t'+s+t'$中出现至少3个t', 我们递归地证明即可。 □

有了Theorem A.1,我们只需要判断全0串或者全1串是否合法即可。可以使用KMP算法或者哈希计算字符串匹配。时间复杂度$O(Tn)$。

B. Bobo String Count

不妨设s的匹配位置分别为$p_1, p_2, ..., p_k$，由于要保证其它位置不匹配，我们要对两个相邻匹配位置间隔$p_{i+1}-p_i$，以及开头/结尾匹配位置p_1, p_k对应的方案数计数。

我们令字符串是0-index的。令$m=|t|$，g_i表示$s[0, i+m-1]$中开头和结尾均匹配的方案数，那么有

$$g_i = \begin{cases} \mathbb{1}\left[s[0,m-i-1]\text{是}s\text{的border}\right] & 1 \le i < m \\ 2^{i-m} & i \ge m \end{cases}.$$

令h_i表示$s[0, i+m-1]$中开头和结尾匹配，且中间其它位置均不匹配的方案数，那么有

$$h_i = g_i - \sum_{j=1}^{i-1} h_j g_{i-j}$$

令p_i, q_i分别表示$s[0, i+m-1]$中开头/结尾匹配，且其它位置均不匹配的方案数，那么有

$$p_i = q_i = 2^i - \sum_{j=1}^{i} h_j \cdot 2^{i-j}$$

令$H(x)=\sum_i h_i x^i, P(x)=\sum_i p_i x^i, Q(x)=\sum_i q_i x^i$，答案为多项式$H^{k-1}PQ$的$x^{n-m}$次系数。$H(x)$的系数可以用分治FFT得到，总时间复杂度：$O(n\log^2 n)$。

C. Best Carry Player

题目可以转化为平面上有若干圆，在半径$r=r_1+r_2$内找到一个点使得被覆盖的圆权值和最大。

最优答案一定可以"平移"到某个圆的边界（注意可能在半径为r的圆内无法移动到任意一个圆的边界的corner case，此时的答案即"原地"的答案）。

枚举答案在某个圆上，考虑其它圆对它的贡献是该圆上的一段区间，半径r的限制也对应在该圆上的一段区间，排序扫描找到贡献最大的位置。复杂度$O(n^2\log n)$。

D. Classical Problem?

不妨假设$K < n$。

令L为待定常数，$f_{d,a,b}$为方程$ax^d + bx^{d-1} \le n$的最大正整数解，$B = \lceil \max(n^{1/L}, \log_2 n + 1) \rceil$，我们断言最优进制一定属于下述集合

$$\left(\{f_{d,a,b} \mid 1 \le d \le L, 1 \le a + b \le \log_2 n, a > 0\} \cup \{2, 3, \ldots, B\}\right) \cap \{2, 3, \ldots, K\}.$$

若不然，则存在最优进制$k > B$但不属于上述集合。令$(a_0, a_1, \ldots, a_\ell)$为$n$在进制$k$下的表示，即$\sum_{j=0}^{\ell} a_j k^j = n$。由$B$的定义可知$1 \le \ell < L$。我们下面断言，$a_\ell x^\ell + a_{\ell-1} x^{\ell-1} \le n$的最大正整数解恰为$k$。有了这个断言，我们就立刻得到矛盾。

我们再次使用反证法证明上述断言。若不然，则$a_\ell (k+1)^\ell + a_{\ell-1}(k+1)^{\ell-1} \le n$。注意到

$$a_\ell (k+1)^\ell + a_{\ell-1}(k+1)^{\ell-1} \ge a_\ell k^\ell + a_{\ell-1} k^{\ell-1} + k^{\ell-1} \ge a_\ell k^\ell + a_{\ell-1} k^{\ell-1} + (\lceil \log_2 n \rceil + 1) k^{\ell-1} > \sum_{j=0}^{\ell} a_j k^j = n.$$

这就得到了矛盾。

我们取$L = \log_2 n$就能得到一个$\mathrm{polylog}(n)$时间的算法。在本题中取$L = 9, 10, 11, 12$算法运行时间较好。

E. Double it or Give it to the Next Person

E.1. 最优策略. 在这一部分，我们先说明每位选手的最优策略是怎样的。定义$f(x) \triangleq \lfloor \frac{x-1}{d} \rfloor$为从$x$枚硬币开始能Pass的轮数,我们将证明以下定理:

Theorem E.1. 令先手所拥有的硬币数为x,后手所拥有的硬币数为y,那么先手的一种最优策略如下:

(1) 如果$x \le y$,那么执行*Double*操作;
(2) 否则如果$f(x) > f(y)$,那么执行*Pass*操作;
(3) 否则$x > y$且$f(x) = f(y)$:
  (a) 如果$x \le d$，那么执行*Double*操作;
  (b) 否则执行*Pass*操作。

注意由$f(x) > f(y)$可以得到$x > y$,故Theorem E.1中包含了所有情况。

我们将Theorem E.1-(2)重写成以下引理并证明:

Lemma E.2. 令先手所拥有的硬币数为x,后手所拥有的硬币数为y,如果$f(x) > f(y)$,那么先手的唯一最优策略为执行*Pass*操作。

Proof. 如果$f(x) > f(y)$时先手执行Double操作,那么先手会以$1-p$的概率直接输掉，即先手获胜概率$< p$。否则先手可以一直Pass这样后手必须先Double;此时后手会以$1-p$的概率直接输掉，即先手获胜概率$> 1-p$。由于$p < \frac{1}{2}$，此时先手执行Pass操作显然更优。 □

证明Theorem E.1-(1)以及Theorem E.1-(3)的最优性要相对麻烦一点。我们先考虑证明Theorem E.1-(1)。令$P(x, y)$表示先手/后手所拥有的硬币数分别为x/y时双方最优策略下先手的胜率,我们先给出以下引理:

Lemma E.3. 假如当先手所拥有的硬币数为x,后手所拥有的硬币数为y,如果$x \le y$且$2x \ge y$，那么$P(x, y) = \frac{p}{1+p}$,且双方的一种最优策略是轮流执行*Double*操作。

Proof. 不难算出双方轮流执行Double操作时先手获胜概率为$\frac{p}{1+p}$。我们接下来只需要证明$P(x, y) = \frac{p}{1+p}$即可。考虑后手每一轮复制上一次先手的策略，这样由于$x \le y$; Double操作一定是双方交替执行，因此我们有$P(x, y) \le \frac{p}{1+p}$。再考虑先手执行Double操作，执行成功后在之后的每一轮复制上一次先手的策略，这样由于$2x \ge y$,Double操作也一定是双方交替执行，因此我们有$P(x, y) \ge \frac{p}{1+p}$,故$P(x, y) = \frac{p}{1+p}$，引理得证。 □

我们接下来需要考虑$x \le y$且$2x < y$的情况:

Lemma E.4. 假如当先手所拥有的硬币数为x,后手所拥有的硬币数为y,如果$x \le y$且$2x < y$，那么此时先手的一种最优策略是执行*Double*操作。

在证明Lemma E.4之前，我们需要以下引理:

Lemma E.5. 对于任意整数$x, y, k \ge 0$,如果$f(x) > f(y), x - kd \ge 0, y - kd \ge 0$,那么有$P(x, y) \le P(x - kd, y - kd)$。

Proof. 根据Lemma E.2以及$f(x) > f(y)$，先手必须执行Pass操作，且后手在之后连续执行Pass操作后,先手都必须使用Pass操作来回应。因此根据$P(\cdot, \cdot)$的定义得证。 □

我们现在可以证明Lemma E.4了。

Proof of Lemma E.4. 首先由于$2x < y$，根据Lemma E.2,先手立即执行Double成功后后手必须使用Pass操作来回应;我们假设先手不立即执行Double操作而是选择先连续执行k次Pass操作后再执行Double操作，那么根据Lemma E.2后手都必须使用Pass操作来回应，且此时先手的胜率为$p \cdot (1 - P(y - kd, 2(x - kd))$,而先手立即执行Double操作后的胜率为$p \cdot (1 - P(y, 2x))$, 我们接下来只需要证明对于任意$k \ge 1$都有$P(y, 2x) \le P(y - kd, 2(x - kd))$即可。

由于$2x < y$,所以有$f(y) \ge f(2x)$。我们分两种情况讨论:

- 如果$f(y) = f(2x)$,那么$P(y, 2x) < p$，而根据$y > 2x$有$f(y - kd) > f(2(x - kd))$,故$P(y - kd, 2(x - kd) > 1 - p$。再根据$p < \frac{1}{2}$可以得到$P(y, 2x) \le P(y - kd, 2(x - kd))$。
- 如果$f(y) > f(2x)$，那么有

$$P(y - kd, 2(x - kd)) \ge P(y - kd, 2x - kd) \ge P(y, 2x),$$

其中第一个不等号是根据对方硬币变少时可以copy硬币不变时的操作得到的，第二个不等号是根据$2x < y$以及Lemma E.5得到的。

□

结合Lemma E.3以及Lemma E.4,我们证明了Theorem E.1-(1)。我们最后将Theorem E.1-(3)重写成以下引理并证明:

Lemma E.6. 令先手所拥有的硬币数为x,后手所拥有的硬币数为y,如果$x > y$且$f(x) = f(y)$,那么先手的一种最优策略为:

(1) 如果$x \le d$，那么执行*Double*操作;
(2) 否则执行*Pass*操作。

Proof. 当$x \le d$时先手需要执行Double操作是显然的,我们剩下只需要证明当$x > d, x > y$且$f(x) = f(y)$时先手的一种最优策略是执行Pass操作。当$x > d, x > y$且$f(x) = f(y)$时，令$k = \min\{p \mid p \ge 1, f(2^p \cdot x) > f(2^p \cdot y)\}$，考虑后手方采取以下策略：复制先手方的策略直到先手方进行了k次Double。上述策略一定是可行的，因为不妨设先手方在分别在前k次Double之前进行了$m_1, m_2, ..., m_k$次pass操作。在先手方进行第$j (j < k)$次Double后，两人硬币数分别为$x' = 2^j \cdot x - \left(\sum_{i=1}^{j} 2^{j+1-i} \cdot m_i\right) d, y' = 2^j \cdot y - \left(\sum_{i=1}^{j} 2^{j+1-i} \cdot m_i\right) d$，满足$f(x') = f(y')$，所以后手可以一直等到先手先Double再自己Double。

在双方进行k次Double后，令$s = \sum_{i=1}^{k} 2^{k+1-i} \cdot m_i$，剩余硬币数分别为$x' = 2^k \cdot x - sd, y' = 2^k \cdot y - sd$，此时后手方按照上述策略能得到至少$1 - P(2^k \cdot x - sd, 2^k \cdot y - sd)$的胜率。根据Lemma E.5，由于$f(2^k \cdot x) > f(2^k \cdot y)$，先手方采取“尽可能Pass”策略，此时得到的$s$最大，$P(2^k \cdot x - sd, 2^k \cdot y - sd)$也最优。另外，注意到当先手方采取“尽可能Pass”策略时，后手方也只能采取“复制先手方的策略直到先手方进行了k次Double”这一策略，所以此时的双方策略均达到最优。 □

至此，Theorem E.1得证。

E.2. 基于最优策略的算法. 根据上述策略模拟，一旦在某个时刻，先手/后手方的硬币数x, y满足$x \le y, 2x \ge y$时，双方就会进入”轮流Double“状态。否则双方的状态会进入一个环（这种情况下，环上每个状态中两人钱都不超过$2d$，所以状态只有$O(d^2)$个，实际上环长会更短）。

记忆化搜索/直接暴力模拟找环即可。（注意在两人能Pass轮数恰好相同时，直接模拟双方Pass的复杂度是$O(x)$的，需要加速这部分。）

最终我们可以用一个无穷长的01串表示双方翻倍的顺序，例如01011(01)表示双方翻倍的顺序为：Alice,Bob,Alice,Bob,Bob,之后Alice,Bob交替。这个01串中从左到右第i位如果是1，给Alice的胜率贡献为$(1-p)p^{i-1}$，循环节部分的贡献是等比数列，最终将所有1位的贡献相加即为Alice的总胜率。

F. Election of the King

注意到每次投票的时候只有倾向最左和最右的候选人会被投。我们把候选人按a_i从小到大排序，留存的候选人总是一个区间，每次投票后区间左端点或者右端点出局。决定是谁出局的是中间的人（如果是偶数个人就是偏左的中间的人），因为一个投右边的人左边所有人都投右边，而我们判断谁出局只要看投右边的人数有没有到达一半。时间复杂度瓶颈在排序，为$O(n\log n)$。

G. Famished Felbat

我们只需求解$\sum_{i=1}^{n}\sum_{j=1}^{m}g(a_i+b_j)$，其中$g(x)=\sum_{i=1}^{L}\lfloor\frac{x}{i}\rfloor$. 注意到对于任意的$B\geq\sqrt{x}$，都有$g(x)=\sum_{i=1}^{\min(L,B)}\lfloor\frac{x}{i}\rfloor+\sum_{i=1}^{B}\max(0,\min(\lfloor\frac{x}{i}\rfloor,P)-B)$.

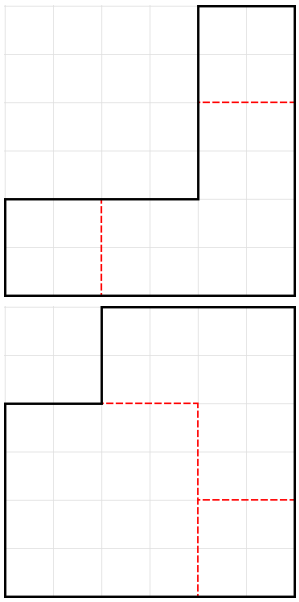
对于每个固定的i，我们首先考虑求解$\sum_{j=1}^{n}\sum_{k=1}^{m}\lfloor\frac{a_j+b_k}{i}\rfloor$。注意到

$$\sum_{j=1}^{n}\sum_{k=1}^{m}\lfloor\frac{a_j+b_k}{i}\rfloor=m\sum_{j=1}^{n}\lfloor\frac{a_j}{i}\rfloor+n\sum_{j=1}^{m}\lfloor\frac{b_j}{i}\rfloor+\sum_{j=1}^{n}\sum_{k=1}^{m}\mathbb{1}\left[a_j \bmod i+b_k \bmod i\geq n\right].$$

对于每一个i，我们可以利用树状数组等数据结构在$O(n\log n)$的时间内求解。另一部分可以类似的进行求解。总体的时间复杂度为$O(n\log n\sqrt{Q})$.

H. Merge the squares!

在一次合并操作中，合并后的大正方形左上角的小正方形是处在一个作为被合并的元素之一的正方形中，合并后的大正方形去掉这个正方形剩下一个L形。我们想要知道这个L形怎么由正方形构成比较好。一个很自然的想法是，对于L形不同的两种形态我们分别如下图划归成两个正方形和一个更小的L形。



我们可以dp算出每个正方形和L形的开销（过程中一次最多合并了多少正方形，以及L形目前由几个正方形构成）以及大正方形的左上角正方形大小是多少的时候最优（对大正方形枚举左上角正方形大小转移，L形则只有一种转移），时间复杂度$O(n^2)$。使用这种做法一次需要最多合并32个正方形。事实上考虑正方形的转移点选在黄金分割已经可以得到一个在$2\log_{\frac{\sqrt{5}+1}{2}}n+c$的下界（$c$代表常数），远远小于50。

另一种做法是把正方形分成对角的两个正方形和两个完全相同的长方形，然后对长方形辗转相减。这个做法也可以$O(n^2)$dp出分割点，一次需要最多合并32个正方形，并且使用黄金分割得到一个在$2\log_{\frac{\sqrt{5}+1}{2}} n + c$的下界。

I. Portal 3

先$O(k)$计算出每对$s, t \in V$在序列中连续出现的次数$c(s,t)$,那么答案就是$\sum_{s,t\in V} c(s,t)d'(s,t)$,其中$d'(s,t)$是加了传送门后的最短路。

通过Floyd-warshall计算出原图最短路$d(\cdot,\cdot)$,那么对一条边(u,v)开传送门对(s,t)的贡献就是

$$c(s,t)\max(0, d(s,t) - d(s,u) - d(v,t), d(s,t) - d(s,v) - d(u,t)).$$

注意到

$$\begin{aligned}&\max(0, d(s,t) - d(s,u) - d(v,t), d(s,t) - d(s,v) - d(u,t))\\ =&\max(0, d(s,t) - d(s,u) - d(v,t)) + \max(0, d(s,t) - d(s,v) - d(u,t)),\end{aligned}$$

这是因为$d(s,t) - d(s,u) - d(v,t)$和$d(s,t) - d(s,v) - d(u,t)$中至多只有一项大于0,不然将两项相加后再移项会违反最短路的三角不等式。

因此可以将贡献分开算。下面考虑统计$c(s,t)\max(0, d(s,t) - d(s,u) - d(v,t))$这一类贡献，固定$s$之后把所有$u$按照$d(s,u)$从小到大排序。枚举$t$和$v$,那么每次对应排序后的一段$u$的前缀加上$-d(s,u)$或者常数，前缀加的位置可以用双指针在均摊$O(1)$时间内找到，答案统计可以用差分。总时间复杂度$O(k + n^3)$。

J. Qu'est-ce Que C'est?

令$dp(i,j)$表示考虑了前i个数且填的数合法，当前的后缀最小值为j的方案数。(这里$j < 0$是可以的，表示最后一个数为负数，且之前没有包含这个数的长度< 0的负和子段)。这里考虑$dp(i,j)$下一个数填k的话(需要满足$-m \le k \le m, j + k \ge 0$)会转移到$dp(i+1, \max(j,0)+k)$，那么从一个$dp(i,j)$的转移就可以看成是一段区间加，用前缀和优化即可在$O(n^2)$的时间内通过。

K. Square Game

题目的要求是求要多少子矩形满足

- 它的上面和下面是红色，左边和右边是蓝色，为了方便叙述我们把这一条件称作合法矩形
- 它呈现的square game 结果是平局，为了方便叙述我们把呈现的square game 结果是红色胜利/蓝色胜利/平局的矩形分别称为红胜矩形/蓝胜矩形/平局矩形

为了方便叙述，下文假设n和m同阶。

这个模型中蕴藏着很多有趣的性质，所以题解中提及了很多最终做法不依赖的性质和无关的做法。如果你只关心这题的$O(n^2)$做法而不关心题目中其余的内容可以略过开头打(*)的段落)。

(*)一个朴素的想法是枚举所有的子矩形，判断它是不是合法的，然后通过bfs、dfs等方式判断它是不是平局矩形。这个做法乍一看是$O(n^6)$的，但是实际上是$O(n^5)$的，这是因为合法矩形至多只有$O(n^3)$个，我们马上就会证明这一点。

(*)我们考虑以(X,Y)（这个记号表示矩形中第X行第Y列的格子，下同）为右下角的合法矩形。我们建立一个二分图，左侧的点代表行，右侧的点代表列。如果有一个左上角为(x,y)右下角为(X,Y)的合法矩形，我们就把代表第x行的点和代表第y列的点连一条边。最终产生的图没有环，这是因为如果存在一个环，不妨设(x,y)是这个环上$x+y$最大的边，那么存在边(x,y')和(x',y)，并且有$x' < x, y' < y$，由合法矩形和边的定义，存在边(x,y')表明矩形中$(x-1,y-1)$是红色，存在边(x',y)表明矩形中$(x-1,y-1)$是蓝色，矛盾。而无环无向简单图中边数小于点数，所以以每个格子为右下角的合法矩形数都是$O(n)$的，合法矩形总数是$O(n^3)$。

(*)观察到上面的做法不够快的一个原因是对每个合法矩形单独地判断它是不是平局矩形，每个矩形的计算结果没有被别的矩形利用到。我们希望尝试了解一个合法矩形的计算结果是怎么帮助其余合法矩形的。

为了处理方便，我们把两个蓝立柱贴在一起中间的缝视为一个宽度为0的蓝胜合法矩形，把两个红横杠贴在一起中间的缝视为一个高度为0的红胜合法矩形。这不会影响平局合法矩形的数量。

我们观察到如果一个合法矩形可以被一个蓝色立柱分开，得到的是两个小合法矩形，并且大矩形呈现的square game 结果可以从小矩形的结果直接得到：大矩形是蓝胜矩形当且仅当两个小矩形都是蓝胜矩形，大矩形是红胜矩形当且仅当两个小矩形中存在红胜矩形，其余情况大矩形都是平局矩形。大矩形被红横杠分开也是类似的情况：大矩形是红胜矩形当且仅当两个小矩形都是红胜矩形，大矩形是蓝胜矩形当且仅当两个小矩形中存在蓝胜矩形，其余情况大矩形都是平局矩形。我们还可以从这个结论推出，上下右边界都确定的合法矩形，按左边界从左到右排序的话，红胜矩形左侧的都是红胜矩形，蓝胜矩形右侧的都是蓝胜矩形。别的三个方向的情况类似。

(*)我们可以利用这一点快速一起计算所有上边和下边高度确定的所有矩形。我们从左到右计算以每个蓝立柱为右边外侧蓝立柱的合法蓝胜矩形和合法平局矩形数量，这可以利用上面的结论快速得到：如果这个蓝立柱和上一个蓝立柱之间是不合法矩形或者红胜矩形，两个值都为0；如果是蓝胜矩形，合法平局矩形数量不变，合法蓝胜矩形数量+1；如果是平局矩形，合法平局矩形数量变成原来平局矩形数量+原来合法蓝胜矩形数量+1，合法蓝胜矩形数变成0。对于两个相邻立柱间的矩形呈现的square game 结果，我们仍然使用bfs、dfs等方式判断。这样我们对所有上边在第x行下边在第y行的子矩形在$O((y-x+1)*n)$的时间内计算出了结果，总时间复杂度$O(n^4)$。

(*)下面会叙述三个不一样的从上面的$O(n^4)$做法优化而来的$O(n^3)$做法。

(*)注意在做上边在第u行下边在第d行的情况时，本来要做的对第l列到第r列之间的矩形的检查实际上就是判断在第$u-1$行至第d行构成的子矩形中，$(d,l-1)$与$(d,r+1)$是否连通，以及对$l \le i \le r$, 是否存在(d,i)是红色并且与第$u-1$行的任何红色格子连通。这只需要知道第d行格子的连通情况以及每个点是否与第$u-1$行的任何红色格子连通。(u,d)的情况可以从$(u,d-1)$的情况$O(n)$用bfs、dfs等方式得到，所以我们只要对u相同的情况按d从小到大做，就可以得到一个$O(n^3)$的做法。

(*)另一个做法是：我们主体还是使用上面$O(n^4)$的做法。但是在准备使用bfs、dfs等方式判断某个矩形呈现的square game 结果时，我们注意到有些时候可以使用以前的结果：当矩形中有一个红色横杠的时候，我们可以用它分成的两个子矩形的结果立即得到这个大矩形的结果（子矩形结果可以通过按$d-u$从小到大枚举的方式提前计算，用存储对确定的上下右边界最左的蓝胜矩形左边界和最左的红不胜矩形左边界来存储和查询）。如果没有红色横杠，那么这个合法矩形没有红色横杠也没有蓝色立柱。我们将要证明这也的矩形的面积和是$O(n^3)$的。这是因为内部没有红色横杠也没有蓝色立柱的合法矩形两两之间关系只有包含和相离，这意味着两个不相离的这样的矩形长加宽的和不一样。我们把这样的矩形按长加宽的和分组，只有$O(n)$组，每组内矩形两两相离，面积和为$O(n^2)$，所以总面积和$O(n^3)$，所以我们用bfs、dfs等方式造成的总开销是$O(n^3)$的，总时间复杂度也为$O(n^3)$。

(*)最后一种要介绍的$O(n^3)$做法用到了一种全新的判断一个合法矩形呈现的square game 结果的方法。

现在让我们聚焦于一个合法矩形。我们现在关心所有异色格子之间的边（我们不关心同色格子之间的）。我们给这样的边一个方向形成一个向量，使得向量左侧是红色格子、右侧是蓝色格子（左右侧是相对向量的方向来说的）。我们观察矩形的边界，在边上外边没有向量、内部没有向量指向外边，而左上角和右下角外面各有一个指向矩形的向量，右上角和左下角外面各有一个远离矩形的向量。现在我们观察到，每个点和偶数个向量相邻，并且绕一周看指向这个点的向量和从这个点出发的向量相间。现在我们定义模式一是所有向量的后继定义为从它指向的点的视角来看点相邻的向量里这个向量顺时针的下一个向量，模式二是所有向量的后继定义为从它指向的点的视角来看点相邻的向量里这个向量逆时针的下一个向量。无论在模式一还是模式二下，这些向量都只形成环和链，因为除了角上的四个向量都有恰好一个前驱一个后继。而唯二的两条链分别从左上角和右下角出发到达右上角和左下角（顺序不一定）。下面我们将说明，在模式一下，左上角出发的链到达右上角（而不是左下角）等价于红不胜，并且相似地有在模式二下左上角出发的链到达右上角（而不是左下角）等价于蓝胜。

在模式一下，一个向量和它后继的向量左侧的红色格子在同一个四连通块中。左上角的向量左侧的红色格子在矩形上边的红色连通块中，而左下角的向量左侧的红色格子在矩形下边的红色连通块中，如果左上角出发的链到达左下角意味着上下两个红色连通块在我们给出的图里

是同一个，即存在一条矩形内红色四连通路径连通上下边。而如果如果左上角出发的链到达右上角，我们考虑这条链和外侧上边界构成的封闭图形。这个图形内和图形外的红色块并不四连通，因为向量的两侧不可能同时是红色格子。而矩形上边的红色连通块在这个图形内部、矩形下边的红色连通块在这个图形外部，这表明他们不连通，从而不存在一条矩形内红色四连通路径连通上下边。模式二的情况是对称的。

于是我们可以$O(n^2)$预处理整个矩形里的向量构成的环和链的情况来$O(1)$地判断任意合法矩形呈现的square game 结果。注意作为整个矩形的子矩形的合法矩形和我们聚焦的纯粹合法矩形稍有不同：左上角出发的向量不断走后继可以先在子矩形内到达左下角然后在子矩形外面到达右上角，所以判断左上角出发的矩形内的链在矩形内是到达左下角还是右上角其实是看在大矩形里左上角的向量沿着后继走是先到达左下角还是先到达右上角（也可能只到达一个）。

(*)我们用这个方法代替bfs、dfs等方式就可以把上面的$O(n^4)$做法优化成$O(n^3)$，但是我们有了如此强有力的工具之后可以更进一步达到$O(n^2)$。

每个合法矩形上边的红色横杠是唯一左右极长红色横杠的一部分，我们称这个合法矩形挂在这个左右极长红色横杠上。所有左右极长红色横杠的长度和是$O(n^2)$的，因为每个红色格子至多在一个左右极长红色横杠中。我们考虑对每个左右极长红色横杠，计算挂在它下面的合法且平局矩形数量。左右极长红色横杠的下方贴着一些上下极长蓝色立柱（我们下面把这些称作实柱），在左右两侧还各可能有一个上下极长蓝色立柱从左/右侧贴着左右极长红色横杠（我们下面把这些称作虚柱）。上下极长蓝色立柱的长度和也是$O(n^2)$的，并且每个上下极长蓝色立柱至多作为一个左右极长红色横杠的实柱。挂在我们关注的左右极长红色横杠下方的合法矩形的左右外侧都是一个实柱或者虚柱的一部分。我们把这些矩形分成三类：被两根虚柱夹着的矩形、被两根实柱夹着的矩形以及被一根实柱和一根虚柱夹住的矩形。

对于被一根实柱和一根虚柱夹住的矩形，我们可以枚举夹它的是哪根实柱和哪根虚柱，以及它的高度（这不超过实柱的高度），然后$O(1)$计算这个合法矩形是不是平局。复杂度O(实柱长度和)，总复杂度$O(n^2)$。

对于被两根虚柱夹住的矩形，我们注意到它的下边只有两种情形：下边内侧高度与比较浅的虚柱一样深，或者下边是左右极长红色横杠。我们从下往上枚举左右极长红色横杠，每枚举到一个从第l列到第r列的被两个虚柱夹住的左右极长红色横杠都计算出以它为上边的外边、左右两边内侧分别是第l列和第r列的合法矩形中，有多少个红胜矩形和多少个平局矩形。这两个值我们可以通过上一个(l,r)的值$O(1)$的计算出来：如果左右虚柱不夹着上一个这样的左右极长红色横杠，只有一个合法矩形且为情况一，我们$O(1)$计算；否则考虑计算左右虚柱和这两个左右极长红色横杠围住的矩形：如果是红胜则红胜矩形数相较上一个+1，平局矩形数不变；如果是蓝胜则红胜矩形数和平局矩形数都变成0，如果是平局则平局矩形数变成原来的红胜矩形数+平局矩形数+1，红胜矩形数变成0。总复杂度$O(n^2)$。

对于被两根实柱夹住的矩形，我们从左向右枚举实柱，再从上向下枚举深度（不超过该实柱深度），计算挂着的以被枚举的实柱为右边外侧、枚举的深度为深度的合法矩形的情况。我们用一个单调栈维护实柱，对每一个单调栈内的实柱，记录以它为右边外侧、深度和它的深度相同、以这个左右极长红色横杠为上边外侧的矩形中有多少个是蓝胜矩形、多少个是平局矩形。单调栈中对于深度相同的实柱我们只保留最后一个,合并方法和上一段一致。我们枚举深度的时候一边计算一边弹单调栈顶。如果单调栈顶的深度和枚举的深度一致就用单调栈里存的信息$O(1)$得到平局矩形数量(需要计算一个合并，合并方法和上一段一致，但是不更新在单调栈里只是得到答案)，如果栈顶深度更深就只有一种合法矩形直接计算即可。复杂度O(实柱长度和)，总复杂度$O(n^2)$。

综上所述，我们在$O(n^2)$的时间内计算出了合法平局矩形数量。

L. We are the Lights

由于后执行的操作会覆盖先执行的操作，我们可以考虑从后往前倒推。倒推的过程中我们需要开布尔数组记录哪些行列在之后的操作中被覆盖了，以及当前已经被覆盖的位置中点亮的灯数。时间复杂度为$O(n+m)$。